

most significant bits

News and Information For High-Tech Professionals

Published by Pocket Protector Press™

A division of Stout Systems



Winter 2007

In This Issue

Job Search 2007

Results of a survey conducted by Stout Systems to learn more about job search habits in southeast Michigan IT workers.

Extending Windows Forms Controls

What to do when the supplied control doesn't quite behave as the end users expect.

Recent News

Stout Systems welcomes new employees Madhuri Joginpally and Robert Baldwin (Quality Assurance Engineers) and Steven Thebo (Software Engineer).

Job Openings

Check out current permanent and contract openings on our Web site StoutSystems.com.

Current Candidates

A sampling of candidates we represent is available on StoutSystems.com.

Subscribe To Our Newsletter

If you would like to receive this newsletter directly at your desk or in-box, send a note to info@stoutsystems.com providing your subscription info (e-mail or ground mail address).

Call us:

(734) 663-0877

What do you do to find a job?

Job Search 2007

by John Stout

We are seeing a continued demand for IT workers, both for permanent hire and contract. The most significant change since the New Year is that there are more prospective candidates to do the work.

We recently sent a survey to numerous IT job seekers regarding their usual means of finding employment in the industry. While the results of individual surveys are confidential, I thought you would like to know the overall results.

Of course the responses only reflect opinion—and mostly the opinions of those in southeast Michigan where we are headquartered. However, these opinions are the perceptions through which local IT employees conduct their job searches.

As for the demographics, 50% of the respondents have been in the industry 10-25 years, and an additional 25% over 25 years. About half are software developers, 25% are in project management, with the rest reporting job titles that include tech writer, analyst, systems administration and software quality assurance.

Below are the survey questions and a summary of the answers.

What sources do you count on to provide you with job openings? (Note that many people gave multiple answers to this question.)

The primary answer to this question was—no surprise—"on-line job boards," which was the method of choice of 86%



of the respondents. "Word of mouth" and "networking" together appeared in 58% of the responses. Our respondents rely on one another and their industry contacts. Interestingly, only 14% of the responses included "recruiters" as a source they can count on, although undoubtedly many of jobs they listed in the on-line boards are from recruiting organizations.

What is the one most important factor that attracts you to a specific job opening?

The most common answer was "the technologies advertised" at 31%. "Location" was a close second at 28%, beating out "salary" which came in at only 14%.

If you use the Web for job searches, where do you look most often for high tech job openings?

Not much surprise here as Dice and Monster led by 3 to 1 over any of the many other job boards named. (Thanks to the 12% of you who answered "Stout Systems".)

continued on page 4

Extending Windows Forms Controls

by K. Alan Robbins

Microsoft's .NET development environment comes with an extensive library of input controls. While these controls offer considerable features out of the box, there are cases where the default functionality does not meet the requirements of the solution being developed.

The most common scenario is that the supplied control doesn't quite behave as the end users expect. A user who primarily uses Excel, for example, is used to a specific set of editing keys when working with grid data. Perhaps a key function, such as CTRL+Shift element selection, is not provided by the control.

Another case is when a different graphical representation is desired in order to make the user experience better, such as a list box that displays both graphical and text elements.

In rare cases the information being displayed simply doesn't fit into any of the existing controls and an entirely new control needs to be created from scratch.

Extending and modifying the functionality of a control is an area where many things are easy to do. There are, however, modifications that appear very simple on the surface but end up being extraordinarily complex, requiring intimate knowledge of Windows internals. Unless developers have experience creating and/or modifying controls, the lowest risk choice is simply to accept the way they work out of the box. In general, the difficulty of extending a control is a function of the control's complexity. A TextBox is a pretty

basic control and is therefore fairly straightforward to extend. The DataGridView is remarkably complex.

Derived Controls

Taking an existing control and adding a small piece of functionality is generally done by creating a new class that inherits from an existing control. A common control extension is to give a control affinity to a business object that the control represents, because your business objects don't lend themselves to data binding. A class that inherits from a control is known as a *derived control*. You can cast the control to your known type in the various .NET events such as Click or Select handlers.

If a derived control is placed on the designer surface, it will need to behave properly or you'll get the dreaded Red Screen of Death, or RSOD, when you switch to design view. Any property needed to instantiate the control at design time—such as the constructor, location, size, text, etc.—will be called by the designer when you open the form or user control in design mode; if these properties cannot be resolved at design time the RSOD is the result. The easiest way to avoid the RSOD is to use a setup method as opposed to a custom constructor and not to override properties needed at design time in unusual ways.

Another approach is to programmatically arrange controls within a container control at run time. The most common implementation of this technique is Microsoft Outlook™. Outlook's main user interface paradigm is a scrollable panel that contains a collection of panels that expand/collapse and render within the parent container using dock/fill properties.

The most common container for these types of controls is the panel. You can override the panel's IContainer interface to implement custom scrolling behavior and draw directly on the panel's surface to add visual elements.

Third Party Controls

There is a thriving market in third party controls. Before buying a control or control suite you should ask the following questions:

- Do you need agility with respect to upgrades? Adding a control library will complicate the process of upgrading to the inevitable next version of .NET.
- Do you need most of the functionality the control offers? It may be more cost effective to add the one feature you need to an existing control than to purchase an entire library.
- Do you have confidence that the vendor will be around in the future? Alternatively, is source code available?

There are literally thousands of control extensions and controls available for download at no cost, almost all of which come with source code. Before using one of these open source controls in your mission critical application, ask yourself the following questions:

- Do you understand how the control works? Just because the price is right doesn't mean it was done properly.
- Is there a thriving, active developer community involved with the project? This insures that you can get questions answered quickly.
- Are there licensing requirements? Many open source controls require that you credit the developer in your EULA or **About** box.

Owner-Drawn Controls

At the next level of sophistication are controls that paint themselves (owner-drawn controls), or a combination of a container control with user-defined painting added. Once you become familiar with the details of handling the paint event—and the meaning of the various control style bits—the main challenge involved with creating these controls comes in getting everything to line up properly.

Most monitors today support a resolution of 96 dots per inch (dpi) and 120 dpi. The latest flat panel displays can go as high as 177 dpi. All drawing, regardless of the hardware, comes down to a pixel—the smallest unit on the device that can be colored. As Windows evolved and hardware capabilities expanded, different schemes for reconciling the size of an object “as designed” versus the size of an object “as drawn” were conceived. In order to maintain a semblance of compatibility with existing applications, Microsoft implemented layers of abstraction on top of the core drawing APIs. .NET introduced GDI+. The latest abstraction is the Windows Presentation Foundation (WPF). Over the intervening years a number of different measurement systems have been introduced ranging from points to twips to inches.

GDI+, introduced with .NET 1.0, uses a floating point coordinate system. The Windows APIs, however, still draw to the screen using integers. Aliasing and smoothing algorithms in GDI+ convert these floating point values into pixel values with varying degrees of success. When mixing GDI+ measurements with older functions (such as the mouse position, which is an integer), the secret is to round up the final result using one of the Ceiling functions.

A picture box has been around since the earliest days of Windows, so it sizes in pixels. If you have an image that you want to put in a picture box, the size property for the picture box at 96 dpi screen resolution exactly matches the size in pixels required for an exact fit in a drawing tool such as Photoshop. If you change the screen resolution to 120 dpi, you'll need a different sized picture to make it fit in the box properly, or you'll have to load the picture at run time and size it yourself. This is how everything worked in the good old days of Windows 1.0.

The image list, which appeared with .NET 1.0, uses GDI+ measurement units which are 1/100th of an inch—almost 96 dpi but not quite. Measure String returns a floating point number that is padded for glyphs, but you can't paint

.00004 pixels. The TextRenderer class, new for 2005, returns string sizes in pixels that are not padded. You'll find the built-in help terribly lacking in these details. These kinds of inconsistencies result in the sizing challenge being more black art than science.

Controls that support AutoSize mode will report one size in the form's constructor and another size in the form's load event—and that size will change as the form is resized. In this case all of your custom layout code belongs in the form's Resize event.

A common mistake is to apply a little fudge factor to get things to look right. This works fine until the target system doesn't have the same display properties as the development system does. If you find that you just can't get things to line up properly without adding a fudge factor, you're probably mixing measurement units between different generations of controls, or doing the rounding at the wrong point.

If you don't need to draw lines, circles, or other graphic elements on your control you should strive to use the Docking/AutoSize functionality and let the Windows Forms Engine handle the layout calculations for you. Trying to mix docking and auto sizing with programmatically drawn controls

and/or graphics is far more complicated than doing all of the resizing yourself.

Summary

.NET has made it very easy to do so many things, but when you dive deep into the inner workings of a Windows Forms application the underlying complexity can be intimidating to say the least. Armed with enough knowledge, you can create forms and controls completely from scratch that behave exactly as you want them to. The real question is whether or not the time and effort required is justified by the need. If you've never ventured into this area, proper estimation of the effort is extremely difficult and you should make sure that the project manager is aware of the risk involved.

K. Alan Robbins is a Senior Software Architect with Stout Systems. He has been architecting successful solutions for customers of all sizes for over 25 years, as well as managing teams of developers. When not writing code or project plans he is an active amateur radio operator and boater. He can be reached at alanrobbins@stoutsystems.com.



continued from page 1

Do you have any concerns about your ability to find the job you want? If so, what are your concerns?

Very interesting responses, in that the most common answer was that there were no concerns, expressing a fairly high confidence in people's ability to find the job of choice. Where concerns were expressed, "lack of jobs" came up as the most common answer but at only 12%.

One other point of note: about 5% of the answers to this last question included "age discrimination". With the rising age level of our profession, it will be interesting to see if this concern increases in a future survey.

John W. Stout is the founder and president of Stout Systems Development. He has nearly thirty year's experience in the software industry. He is also sought after as a technology speaker, presenting sessions at developer conferences and user groups.



ADDRESS SERVICE REQUESTED

Presorted Standard
US Postage Paid
Permit #187
Ann Arbor MI

In This Issue:

- Job Search 2007
- Extending Windows Forms Controls